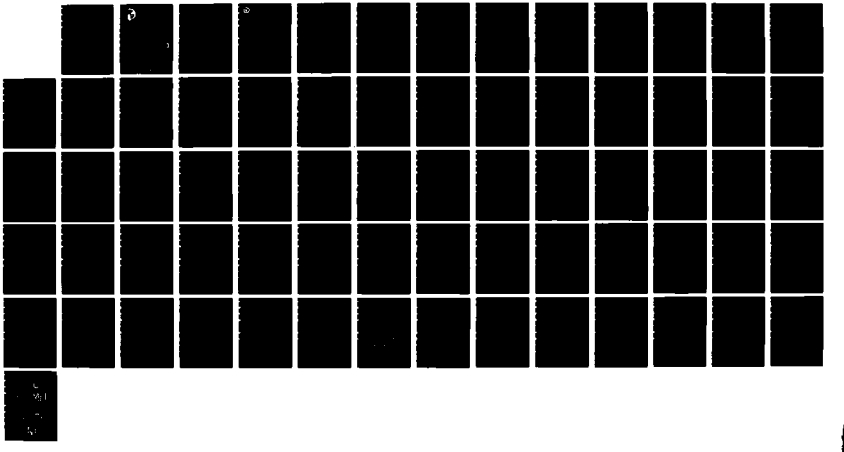
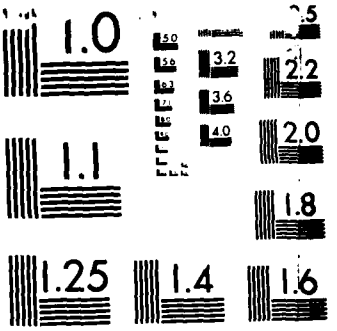
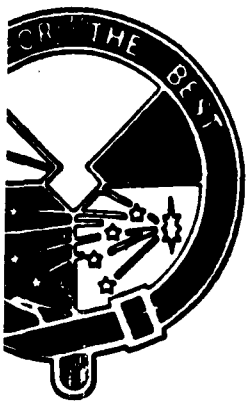
AD-A165 387 SOFTWARE TESTING OF MULTIPROCESSOR SYSTEMS(U) ARMY ELECTRONIC PROVING GROUND FORT HUACHUCA AZ E L ANDERSON OCT 85 1/1

UNCLASSIFIED F/G 9/2 NL







(2)

UNCLASSIFIED

DDC AD NUMBER _____

FUNDING PROJECT NO. IT665702D625

TECOM PROJECT (TRMS) NUMBER 7-CO-R85-EPO-006

TEST ACTIVITY REPORT NO. _____

TEST SPONSOR: U.S. ARMY TEST AND EVALUATION COMMAND

METHODOLOGY INVESTIGATION

FINAL REPORT

SOFTWARE TESTING OF MULTIPROCESSOR SYSTEMS

by

Edward L. Anderson

October 1985

U.S. ARMY ELECTRONIC PROVING GROUND

FORT HUACHUCA, ARIZONA

UNCLASSIFIED

DISPOSITION INSTRUCTIONS

Destroy this report in accordance with appropriate regulations when no longer needed. Do not return it to the originator.

DISCLAIMER

Information and data contained in this document are based on input available at the time of preparation. Because the results may be subject to change, this document should not be construed to represent the official position of the U.S. Army Materiel Command unless so stated.

The use of trade names in this report does not constitute an official indorsement or approval of the use of such commercial hardware or software. This report may not be cited for purposes of advertisement.

DEPARTMENT OF THE ARMY
HEADQUARTERS, U.S. ARMY TEST AND EVALUATION COMMAND
ABERDEEN PROVING GROUND, MARYLAND 21005-5055

REPLY TO
ATTENTION OF

AMSTE-TC-M

2 0 FEB 1986

SUBJECT: Methodology Investigation Final Report, Software Testing of Multiprocessor Systems, TECOM Project No. 7-CO-R85-EPO-006

Commander
U.S. Army Electronic Proving Ground
ATTN: STEEP-TM-TO
Fort Huachuca, AZ 85613-7110

1. Subject report is approved.

2. Test for the Best.

FOR THE COMMANDER:

Grover H. Shelton

GROVER H. SHELTON
C, Meth Imprv Div
Technology Directorate

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER TRMS No. 7-CO-R85-EPO-006	2. GOVT ACCESSION NO. ADA165387	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) METHODOLOGY INVESTIGATION FINAL REPORT--SOFTWARE TESTING OF MULTIPROCESSOR SYSTEMS		5. TYPE OF REPORT & PERIOD COVERED Methodology Investigation Final Report 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Edward L. Anderson		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS U.S. Army Electronic Proving Ground Fort Huachuca, AZ 85613-7110		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS IT665702D625
11. CONTROLLING OFFICE NAME AND ADDRESS U.S. Army Test and Evaluation Command Attn: AMSTE-TC-M Aberdeen Proving Ground, MD 21005-5055		12. REPORT DATE October 1985 13. NUMBER OF PAGES 63
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Software Measurement
Hardware Monitor
Software Monitor

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The Software Testing of Multiprocessor Systems methodology investigation established methods for evaluating software or systems containing multiple interactive processors. A number of multiprocessor monitors/debuggers were examined for function and design approach. The investigation resulted in the development of a hybrid monitor concept: the integration of both hardware and software monitor capabilities into a central monitor system. This phase of the investigation resulted in revision of the specification documents and continued design and implementation of the prototype hybrid monitor.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

FOREWORD

Ultrasystems Defense and Space Systems, Incorporated, Sierra Vista, Arizona assisted in the preparation of this document under Contract Number DAEA18-83-C-0003.

1. SUMMARY

1.1 Background

The increasing complexity of battlefield automated systems (BAS) has dictated the need to use multiple processors to enhance system reliability and to provide timely processing in systems that process large volumes of data. One such system, Position Location Reporting System (PLRS), has completed developmental testing; several more (Remotely Piloted Vehicle (RPV), Facility Intrusion Detection System (FIDS), and All Source Analysis System (ASAS)) are in the development phase. Additionally, the trend is toward distributive processing systems which will have multiple interactive processors and common software functions. Very little has been done to establish comprehensive test procedures for evaluating these systems, particularly under full load conditions.

1.2 Objective

The objective of this investigation was to establish methods and procedures for evaluation of software for systems containing multiple interactive processors.

1.3 Summary of Procedures

1.3.1 Previous Efforts

a. The methodology investigation for Software Testing of Multiprocessor Systems (STMS) evolved from that outlined in appendix A. The investigation began with an analysis of methodologies used by other military departments, government agencies, and private industry for software testing of multiprocessor systems. Hardware monitors (logic analyzers) were subsequently investigated in more detail and a hardware monitor trade-off study performed. Because measures of performance (MOPs) suitable for multiprocessor systems were not discovered in the literature, a preliminary set of MOPs was developed. The concept of the hybrid monitor (HM), which evolved from this investigation, was evaluated and documented.

b. The HM concept (an integration of both hardware and software monitors into a combined monitoring system) evolved from the investigation of standalone hardware and software monitors. The concept was analyzed to identify possible test configurations and to determine the general applicability to multiprocessor system testing. Efforts were initiated to develop the concept into a prototype tool to be used for refining and validating the methods and procedures for multiprocessor software testing. These efforts included the selection of a hardware monitor and the top-level design and implementation of selected operator interface functions for the prototype HM.

1.3.2 Current Efforts

a. The most recent phase of the investigation included refining the requirements specifications to incorporate the Tektronix DAS 9100 as the logic analyzer component of the HM's Hardware Monitor Function. Design and implementation of the central controller software continued to the extent allowed by available resources.

b. The initial goals to complete a prototype HM and validate the methodology on a tactical system relied upon anticipated support from system developers. Although project personnel from both PLRS and RPV expressed interest in the HM, the available resources and system availability were insufficient to support this phase of the investigation. As a result, the proposed tasks of preparing and validating test methods and procedures on a tactical system were not feasible.

1.4 Summary of Results

a. Investigation of methodologies used by others for software testing of multiprocessor systems revealed that none of the approaches provided a general purpose capability suited for the intended environment. The examination did serve to identify three approaches commonly used: software monitors, communication link monitors, and hardware performance monitors. Hardware monitors were studied most thoroughly, representing the most mature technique which met the test needs of the multiprocessor environment. This activity culminated in a detailed trade-off study of the hardware monitors appropriate to the STMS investigation.

b. Additional activities at this stage of the investigation resulted in the development of MOPs, existing multiprocessor performance measures being nonexistent. The proposed MOPs were further refined as the concept of the HM evolved.

c. Evolution of the HM proceeded from an initial test methodology to development of preliminary requirements specifications. These specifications were further expanded and development of software for a prototype HM was undertaken. The results of these early stages of the investigation are summarized in section 2 and detailed in the previous methodology report, dated October 1984. A-level and B-level specifications are available upon request.

d. Continued development of the hardware monitor functions of the HM has resulted in a prototype tool with nearly complete hardware monitor capabilities. The requirements documents have been updated to allow use of the available hardware monitor, and the associated central controller software has been designed and implemented. Only the Tektronix DAS 9100-specific processing and the host computer interface functions remain to be developed to provide a hardware monitor capability for the HM. The phases in the development of a complete HM--a central controller, hardware monitor, and software monitor and kernels--would provide full prototype capabilities. After selection of an initial target system, the system-specific software kernels would be developed. Completion of the software monitor component of the HM would lead to validation of the concept by application of the prototype HM to testing of a tactical system.

1.5 Analysis

Although the existing hardware monitor component may be used in a stand-alone mode, validation of the HM concept would require development of a minimal configuration HM. The application of the HM to a tactical system would serve to validate the HM concept and proposed MOPs. Achieving at least partial success in this effort should be low risk because of the HM's basis on proven software/hardware monitor methodologies.

1.6 Conclusions

a. This investigation has revealed that there are no general purpose multiprocessor monitors for software testing available today. Although both hardware and software monitors were found to be used in testing, neither provides an optimum, flexible monitoring capability suited for a multiprocessor environment. The synergism resulting from integrating the capabilities of both hardware and software monitors--the HM approach--exploits the desirable features of both monitors while minimizing their shortcomings.

b. A prototype HM was partially developed, but is not sufficiently complete to function in a hybrid configuration. Further investigation would allow multiprocessor hardware monitoring, followed by a more extensive effort to provide a software monitoring capability. At this stage, hybrid configurations would be possible for refining and validating the methods and measures for multiprocessor software testing of a tactical system.

1.7 Recommendations

As a result of this investigation, the following recommendations are made:

a. Monitoring of emerging methodologies for software testing of multiprocessor systems should continue. In particular, new developments in hardware and software monitoring and MOPs should be examined with respect to the proposed HM methodology. Refinements to the HM design and proposed MOPs may be expected from appropriate technological advances.

b. The remainder of the hardware monitor component of the prototype HM should be developed and demonstrated. A target system should then be chosen to allow development of the prototype software monitor component. This would be followed by a demonstration of hardware, software, and hybrid configurations. Various MOPs proposed earlier in the investigation should be used and evaluated for future applications of the HM.

c. Pending successful completion of the prototype effort, a methodology investigation should be established to develop a full-scale HM. The central controller function should reside in a Test Item Simulator (TIS), a test driver developed at USAEPG. This proposed investigation would result in a validated methodology as well as refinement of the HM architecture and associated MOPs.

2. DETAILS OF INVESTIGATION

The STMS investigation examined methodologies used by others, evaluated hardware monitors, and developed MOPs and the HM concept. Results of these efforts are summarized below and in the appendices to provide a point of reference for current activities. Details were supplied in the previous methodology report, dated October 1984, and in HM system documentation (available upon request).

2.1 Investigation of Other Methodologies

A number of different multiprocessor monitors/debuggers used on systems were examined for function and approach. The various approaches included the use of software monitors (self-contained or external) and hardware monitors. These approaches are described below:

a. Software monitor. Software kernels are embedded in the target system to collect information of interest. Operator interface, control, and data storage are provided by a host computer connected to the target processor(s). This is an intrusive approach because the software kernel uses resources (CPU, memory, I/O) of the target processor. Software monitors are suitable for single or multiprocessor monitoring.

b. Communication link monitor. This monitor consists of monitoring hardware, data analysis software, and an optional message traffic generator. Communication links between loosely coupled processors (shared memory or common bus architectures are excluded from this category) are monitored/stimulated to collect data on the information transfer on the link. Data are restricted to that which is external to the target processors. If message traffic is not generated, this approach is non-intrusive; monitors which generate traffic, called test drivers, are intrusive in that target processor resources are required to handle the traffic.

c. Hardware performance monitor. This monitor is designed to collect and store performance data from single or multiprocessor systems using only non-intrusive hardware techniques. Hardware monitors are discussed further in the following section.

Although each approach provided certain advantages for the particular application, none provided a general purpose, flexible monitoring capability suited for a multiprocessor environment. Hardware monitors provide a non-intrusive capability suitable for microprocessor-based systems; however, they lack the flexibility and complex measurement capability of software monitors. (Further comparison of hardware and software monitors is provided in appendix C.) Communication link monitors are important in the testing of systems, but can only indirectly measure performance parameters internal to the target system.

2.2 Investigation of Hardware Monitors

a. Commercial hardware monitors, also known as logic analyzers, and microprocessor development systems were investigated to determine their utility in the multiprocessor environment. Their ability to handle mini-computers as well as microprocessors was assessed.

b. Commercially available hardware monitors can be divided into categories of emulators, monitors, and combinations thereof. Emulators, which provide extremely flexible control, are available for most popular microprocessors but only a very limited number of minicomputers. Most monitors possess a capability to monitor mini or microcomputer busses and input/output (I/O) lines with pretriggering conditions applied. At least one monitor is equipped with a programmable pattern generator for target system stimulation.

c. The most significant constraints of the hardware monitors studied are buffer memory limitations and the inability to quickly transfer this information to mass storage. These constraints impose an upper limit on the event monitoring rate.

d. Emulators can in some instances overcome some of the difficulties (e.g., read/write memory limitations) encountered in embedding code for software monitors. However, physical constraints sometimes preclude the use of emulators. In addition, emulators for all processors, especially minicomputers, are not always available. In many of these cases, a software monitor must be used to extract the necessary information.

e. Hardware monitors are non-intrusive, requiring no additional memory or processor time of the target system for data extraction. Conditional triggering capabilities permit optimal information gathering while minimizing the collection of non-useful information.

f. A detailed trade-off study was performed and the Tektronix DAS 9100 was chosen as the most appropriate hardware monitor for a prototype system. Appendix D provides the salient characteristics of this device.

2.3 Measures of Performance

a. MOPs applying to multiprocessor systems were proposed. (See appendix E.) The measures were categorized in groups as follows: system control, memory utilization, component usage, software functions, and network communication. Each category contains several MOPs, some of which may apply only to specific system architectures.

b. Possible areas of application of the MOPs are verification/validation testing, integration testing, hardware fault isolation, software debugging, and developmental testing. Most of the MOPs are equally applicable in the testing of single processor systems and may be measured with varying degrees of accuracy by either software or hardware monitoring techniques.

c. The MOP category of network communication could be measured by software/hardware monitors; however, communication link monitors are especially suited to this task. Communication link monitors with stimulation capability (test drivers) provide a means of measuring performance under varying load conditions. The possibility exists for using the different monitors to measure all of the MOPs of interest in a controlled environment provided by a test driver (e.g., TIS).

2.4 Hybrid Monitor

The HM concept was suggested by the examination of other methodologies used for software testing of multiprocessor systems. This concept was developed further by studying the test methodology and tools. An HM design evolved

from this analysis and efforts were initiated to develop a prototype monitor. Further refinement of the design occurred as application of the HM to specific systems was investigated. Appendix F describes the evolutionary development of the HM methodology.

2.4.1 Hybrid Monitor System

The HM system concept was examined by generating preliminary requirements and operational concepts. These were used to analyze the interrelationships among the components, and identify tradeoffs in the design and operation, of an HM system. The preliminary concepts were then refined to produce draft requirements and design documents.

2.4.1.1 Hybrid Monitor Concept

The concept of applying hardware monitors to performance testing is well established and generally understood. Application of software monitors and hybrid monitor configurations, particularly in a multiprocessor environment, is less well understood. In order to understand the design and operational aspects of a hybrid tool, an investigation was performed to identify advantages and disadvantages of software and hardware monitors. The details of this investigation are provided in appendix C and summarized below.

a. The hardware monitor's greatest strength is that it is portable and non-intrusive (i.e., does not require test system resources), and therefore, has no impact on a system under test (SUT) processor's performance. Other strong points are: it can detect and process events, not always accessible to software monitors, at high rates with a high timer resolution; and that it can process external events. The major drawbacks are: its input width (maximum number of inputs) is limited by the number of probes or channels, it sometimes cannot stop the CPU under test, and it is weak in the tracking of complex data structures such as linked lists.

b. A software monitor complements the strengths and weaknesses of the hardware monitor. Its major strengths are that it can track complex data structures, it is easily controlled by external test systems, it has ready access to processor memory, it has the ability to monitor software which is dynamically relocated in memory, and it possesses an input width which is theoretically unlimited. Its major weaknesses are: it impacts the performance of the test system; only data accessible through the processor instructions can be collected; events can only be resolved to the extent provided by the instruction execution rate; and portability is limited to a similar processor/ operating system combination.

c. The goal of a performance monitor should be to detect and report all possible combinations of events at their rate of occurrence without impacting the processors under test. Neither a pure software nor a pure hardware monitor can approach this goal as closely as can a hybrid.

d. As described above, insertion of a software monitor into a SUT processor will always impact that computer's processing. This is true because the software monitor is simply another program within that processor, and therefore requires system resources (e.g., memory, CPU time). Another impact is incurred by the software monitor's communication with a central monitor. This impact usually takes the form of the CPU servicing interrupts from some I/O device. The HM may circumvent this problem by having its hardware element

monitor predefined memory locations through which the software element will communicate with the central monitor, thus reducing communication impact.

e. The multiprocessor hybrid performance monitor has several potential advantages over hardware or software monitors. It will be able to monitor a network of processors simultaneously. The hybrid should also prove to be easily reconfigurable and allow the tester to approach collection of data using several methods which blend the degree of hardware and software monitoring required for a particular system.

2.4.1.2 Hybrid Monitor Description

The multiprocessor HM concept was further investigated to identify the initial components, configuration, interfaces, and transportability requirements of the HM.

2.4.1.2.1 Hybrid Monitor Components

a. A multiprocessor HM is considered to consist of three major components:

(1) Central controller. This is a computer independent of the SUT. This processor serves as the nucleus for SUT monitoring. Both the hardware and software components report the data which they are collecting to the central monitor for:

- Collection/Correlation.
- Storage/Retrieval.
- Real-Time/Post-Analysis.
- Display.

(2) Software kernels. These are data collection/reporting kernels of code which are embedded within the software structure of the SUT. The software kernels collect data which is addressable through the SUT processors' instruction sets.

(3) Hardware monitors. These are devices which extract SUT performance indicators using probes, comparators, and counters to monitor signals. These signals are monitored by attaching probes to SUT computer backplanes, by inserting off-the-shelf bus interface cards, or by using emulator capabilities.

b. These three components work in unison to provide a flexible tool which can be used to extract a wide variety of data from a SUT. In particular, these components allow simultaneous collection of differing data from multiple processors.

2.4.1.2.2 Hybrid Monitor Configurations

a. The HM should be easily reconfigured to extract information from a SUT, using any degree of hardware and software monitoring required. The method of data extraction may be shifted from a hardware- to a software-

oriented manner (or vice versa). Many different configurations may be necessary to observe the required data during the testing of a single system. This demands a tool which is flexible and easily capable of assuming the required configurations.

b. Possible hybrid configurations include:

(1) Hardware Monitor-to-Central Controller. In this configuration, the hardware monitor extracts data and sends it to the central controller without any interaction with the software kernel. This configuration is used if the data to be observed is detectable without complex logic/data correlation required.

(2) Software Kernel-to-Central Controller. In this configuration, the software kernel performs data collection and reporting without any direct interplay with the hardware monitor. This configuration is assumed if the data to be observed requires complex logic and/or data correlation, or if the SUT configuration disallows probing the required test points using the hardware monitor.

(3) Hardware Monitor and Software Kernel-to-Central Controller. In this configuration, the hardware monitor and the software kernel interact to provide and report the required data. Blending of the required amounts of hardware and software monitoring is performed here, drawing on the major advantage of the hybrid. Any degree or combination of hardware and software monitoring is possible. Some combinations of these are:

(a) Strict hardware and software monitoring, the results of which are correlated by the central controller to provide the required performance measure.

(b) Use of the hardware monitor to provide an input and/or output port. This provides a high-speed communications line over which the software kernel may communicate with the central controller.

(c) Use of the software kernel to detect an event which is reported to the central controller using the hardware monitor. Upon event detection, the software kernel reads/writes to a predetermined memory location. This read/write is detected by the hardware monitor and reported to the central controller or acted upon directly by the hardware monitor (e.g., it is used to trigger the initiation of an instruction trace).

(4) Hardware Monitor Stand-Alone. The hardware monitor can be removed from the multiprocessor hybrid and used as a stand-alone tool. The capabilities in this configuration are those inherent in the particular monitor design.

2.4.1.2.3 Hybrid Monitor Interfaces

The HM, as described previously, is itself a network of processors. Both physical and logical interfaces are required to support this network. The physical interfaces are: hardware monitor-to-SUT, central controller-to-hardware monitors, and software kernels (via SUT hardware)-to-central controller. The logical interfaces are: hardware monitor-to-central controller, software kernel-to-central controller, and software kernel-to-hardware

monitor-to-central controller. Note that while the software kernels are embedded within the SUT's program structure, they are not a part of the actual system being tested.

2.4.1.2.4 Hybrid Monitor Transportability

a. The HM is transportable between systems to be tested. Certain portions of the monitor may have to be altered to fit the new test system. These are:

(1) The software kernel may require recoding or alteration if a new computer type or different executive is present in a SUT.

(2) A different hardware emulator or bus interface may be required if a new computer is present in the SUT.

b. The central controller of the hybrid will remain essentially unchanged. The operator interface, data storage/retrieval, component control, data collection/correlation, data analysis, and data display will remain unchanged since these are all driven by standard message packets. The operator is required to define, using a menu-driven display, the contents of any new message packets which will be produced during testing. Another set of tables is produced (again using menu-driven displays) which directs the formatting and display of test data.

c. The operator interface is such that new command files and specialized functions can be easily defined and added to the test control menus.

2.4.1.3 Hybrid Monitor Requirements and Design

The HM description, summarized above, was refined and expanded to provide the requirements for a multiprocessor HM system. Following the generation of an A-level specification, a B-5 specification was created. These specifications have been finalized during the latest phase of the investigation to include the Tektronix DAS 9100 as the logic analyzer component. Complete information is contained in the specification documents.

2.4.2 Prototype Hybrid Monitor

a. Time and resources permitted the initiation of a prototype HM system for validation of the methodology. A hardware monitor trade-off study resulted in the selection of a Tektronix DAS 9100 with 96 channels of data acquisition capability. (Sixteen channels of pattern generation were also obtained to provide a built-in test (BIT) feature and to experiment with the ability of the HM to stimulate the SUT.) The DAS will comprise the hardware monitor component of the prototype HM.

b. A parallel effort resulted in the detailed design and implementation of selected portions of the central controller. Future efforts should complete the software kernel interface for demonstration of a software monitor on a target test system. Also, the DAS will be interfaced to the central controller for a complete HM demonstration and validation.

c. The central controller of the prototype HM is being hosted on a VAX 11/750 configuration. This initial version will be self-contained, relying

only on the operating system and support software for operation. Eventually, the central monitor could be integrated into the TIS, a digital message stimulator for C³I systems.

d. A logical extension to the HM concept which should be investigated next is the use of a communication link monitor as a fourth major component. A communication link monitor with message generation capability (i.e., a test driver such as the TIS) could provide a controlled, repeatable environment for measuring performance with an HM. It is anticipated that this addition to the HM configuration may provide advantages for performance monitoring of SUTs incorporating local area network (LAN) architectures or where stimulation of the SUT is required to produce varying loads from nominal to saturation conditions.

2.4.2.1 Development of Prototype Hybrid Monitor

Effort continued in the design and implementation of selected central controller functions within the resource constraints of the investigation. Portions of the central controller common to both hardware and software monitoring functions had been previously implemented. Current efforts focused on completing the functions required to interface the hardware monitor and central controller. This effort was limited because the hardware monitor and host computer interface were unavailable.

2.4.2.2 Implementation of HM Hardware Monitor Function

Design and implementation of the hardware monitor function involved selected hardware monitor processes, as well as the required extensions to the previously implemented central controller processing. The following processes were implemented:

a. Data Collection, to allow the Central Monitor Function to collect data captured by the Tektronix DAS 9100 Hardware Monitor and input to the VAX 11/750. This processing includes:

(1) Buffer management which allowed initialization and allocation of buffers for test data input to the VAX 11/750.

(2) Queue management which allowed the data collection subfunction to notify the routing subfunction that test data is awaiting further processing.

b. Data Storage, which allowed the Central Monitor Function to store, on disk, data captured by the Tektronix DAS 9100 Hardware Monitor. This processing includes:

(1) Queue management, which allowed coordination of processing between the storage subfunction and other subfunctions.

(2) Command interface, which allowed the routing subfunction to notify the storage subfunction of the receipt of test data.

(3) Buffer management, which allowed the storage subfunction to coordinate buffer control with the data collection subfunction.

(4) File management, which allowed the definition and control of multiple storage files.

(5) Command interface, which allowed the test control subfunction to define files and their storage criteria.

c. Data Display, to provide a coordinated upgrade of the existing central monitor subfunction in parallel with the upgrades to other subfunctions. This processing includes:

(1) Buffer management, which allowed the display subfunction to coordinate buffer control with the data collection subfunction.

(2) Extended display device processing, which allowed output to the printer and to a disk file.

(3) Modified CRT display handling, which allowed viewing the displays on the same device that is used for the control operator interface.

d. Test control, to provide operator interface, processing for controlling the functions outlined above. This includes:

(1) Updates to the existing interface which allowed switching between the test control menu and CRT outputs of the display subfunction.

(2) Storage control operator interface processing, which allowed controlling the storage of test data.

This processing was implemented in FORTRAN to execute on a VAX 11/750 under the VAX/VMS operating system.

2.4.3 Hybrid Monitor Application to Testing

a. Conceptually, the HM approach and MOPs appear to be a viable means of performance testing of multiprocessor systems. However, it was recognized early in the investigation that a target system was required to evaluate the concept, to determine the scope of applicability to testing, and to refine the methodology.

b. Both RPV and PLRS were examined as vehicles to demonstrate the HM system. PLRS was chosen for further investigation as a potential test bed for the HM because of the availability of the system and the previous use of a software monitor. Because a software monitor had been used on PLRS, it is known that sufficient memory to support software kernels and an I/O channel are available. These conditions lower the risk of demonstrating all of the major HM capabilities on the initial application to a tactical system.

2.4.3.1 Rationale for the Hybrid Monitor System

a. A multiprocessor software test capability is easily justified by an examination of the issues involved. First, validation of multiprocessor systems is inherently difficult; indeed, present software testing of real-time uniprocessor systems is largely inadequate. This problem is compounded by the present trend toward more distributed architectures in C^3I systems (e.g., RPV, PLRS, ASAS, and PJH).

b. Adequate testing of software systems has recently received more attention because the cost of software and software maintenance is becoming a critical item in system development and deployment. The cost of software maintenance alone is estimated by some sources as up to 80 percent of the total life-cycle cost. New approaches, such as evolutionary development, emphasize the need for cost-effective, thorough testing because of the number of software versions deployed during the life-cycle.

c. No general purpose, cost-effective means currently exists for testing multiprocessor systems. Portions of the required capability have been provided by specific tools such as the PLRS software monitor. These tools lack the flexibility of a hybrid approach, and in most instances are too system-specific for general applicability.

2.2.3.2 Benefits to Developers/Testers

A general purpose HM system appears to offer a number of advantages to the developer as well as the tester. The following major benefits were recognized.

a. The HM could determine the adequacy of software testing by measuring events associated with test coverage (i.e., the thoroughness of testing). This use of the HM could support a systematic test methodology for ensuring effective testing, in contrast to current ad hoc techniques with undetermined completeness.

b. The HM allows the characterization/evaluation of software using standard MOPs. The HM should provide a practicable method to adequately evaluate multiprocessor as well as uniprocessor systems, illuminating weak areas in system design and implementation.

c. Software maturity could be improved prior to operational testing/deployment, resulting in increased system availability. Problems not otherwise apparent could surface because of the greater insight into the software operation which the HM approach provides.

d. Life-cycle software support modifications should be reduced because of the detection of errors earlier in the life-cycle. The HM would also provide additional information concerning system operating margin (reserve capacity) useful if future expansion or modification is necessary.

APPENDIX A

METHODOLOGY INVESTIGATION PROPOSAL

January 1984

METHODOLOGY INVESTIGATION PROPOSAL

1. TITLE. Software Testing of Multiprocessor Systems

2. CATEGORY. VISTA, DC³I/Software, Interoperability

3. INSTALLATION. U.S. Army Electronic Proving Ground, Fort Huachuca, Arizona 85613

4. PRINCIPAL INVESTIGATOR. Mr. Richard C. Jacques, Software and Automation Branch, STEEP-MT-DA, AUTOVON 879-1957

5. STATEMENT OF THE PROBLEM. Many of the tactical computer systems currently planned or in development contain multiple processors which will have interactive functions and, in some cases, common software functions. Present software test methodology applies principally to systems with single processors or multiple non-interactive processors. During the DT II of PLRS software testing, problems experienced were attributable to interactive computer functions. TECOM currently cannot reliably nor systematically evaluate the software for systems containing multiple interactive processors.

6. BACKGROUND.

a. History. The increasing complexity of automated battlefield systems has dictated the need to use multiple processors to enhance system reliability and to provide timely processing in systems that process large volumes of data. One such system, PLRS, has completed development testing and several more (TCAC, RPV, FIDS, and ASAS) are nearing completion of the development phase. Additionally, the trend is toward distributive processing systems which will have multiple interactive processors and common software functions. Very little has been done to establish comprehensive test procedures for evaluating these systems, particularly under full load conditions.

b. Progress. FY 83 effort under TRMS Project 7-CO-RD3-EP1-002 has included a detailed evaluation of available software analysis programs for application to multiprocessor systems. Candidate tools included Hughes RAID, Navy PCOTES and TESDATA SMART system. The investigation covered their application to the EPG RPV test program and provided PM/contractor/tester interface for front-end support on RPV software development and testing. Both hardware and software monitor capabilities have been analyzed and requirements generated for multiprocessor testing.

7. GOAL. The investigation will establish methods and procedures for evaluation of software for systems containing multiple interactive processors.

8. DESCRIPTION OF INVESTIGATION.

a. This investigation will develop test methods and procedures for software performance testing in systems containing multiple interactive computers. It will cover the full range of testing from single thread tests to testing under full load (saturation) conditions.

b. The U.S. Army Electronic Proving Ground will conduct the investigation as follows:

(1) Research and evaluate methodologies used by other military departments, government agencies, and private industry for software testing of multiprocessor systems, and prepare a report on the results of that survey.

(2) Examine software testing methodologies used by the development contractors for RPV, TCAC(D), and PLRS and the V&V contractors for those systems and select pertinent procedures for application to the overall methodology.

(3) Prepare test methods and procedures for software testing of tactical computer systems containing multiple interactive computers.

(4) Validate the test methods and procedures developed on an operational system.

(5) Prepare draft TOP for evaluation of software in systems containing multiple interactive computers.

c. Investigation Schedule

	SCHEDULE FY 85 (Qtrs)			
MILESTONE/ACTIVITY	1	2	3	4
Finalize test methods and procedures	x	x		
Validate test methods		x	x	
Prepare draft TOP			x	x
Prepare final report				x

d. This investigation will result in new methods and procedures for software testing in multiprocessor systems.

e. Environmental Impact Statement. The execution of this task will not have an adverse impact on the quality of the environment.

f. Health Hazard Statement. No health hazards are anticipated.

9. JUSTIFICATION.

a. Mission and Impact Statements

(1) Association with mission. TECOM and USAEPG are assigned the mission of conducting development testing on a wide range of electronic battlefield systems. These include systems containing multiple interactive processors.

Software Testing of Multiprocessor Systems (cont)

(2) Present Capability, Limitation, Improvement, and Impact on Test Programs if not Performed in the Proposed Fiscal Year. The current software test methodology applies principally to systems with single processors or multiple non-interactive processors. This investigation is needed to develop objective criteria and a method of evaluating software performance of complex tactical computer systems. This methodology is needed in the near term (FY 84-85) to enable testing of ASAS and SIGMA and for future distributive processor systems. If this task is not completed in FY 85, the capability will not be available to support development testing of ASAS or other distributive processing systems scheduled for the FY 86-89 time frame.

b. Dollar Savings. The successful implementation of method and procedures for software testing of multiprocessor systems is expected to reduce the extent of field testing required for these complex electronic systems. This reduction will result from increased efficiency in the conduct of software testing due to use of valid test criteria and methods. The method and procedures developed in this investigation will enable a more structured test and eliminate the need for extended trial and error type testing. The computation of specific dollar savings estimates is not possible at this time.

c. Workload. The methodology will be incorporated into test plan documents and applied during test execution for systems containing multiple interactive computers. Examples of systems anticipated for testing at USAEPG include:

	Test Schedule (FY)					
System	85	86	87	88	89	90
JTIDS			x	x		
MCS	x			x	x	x
RPV	x	x		x	x	
PLRS		x	x			
DTSS			x	x	x	
SHORAD C2			x	x	x	
JINTACCS			x	x	x	x
FISN		x		x	x	
NAVSTAR		x	x	x	x	x
PJH	x	x	x	x		
GPS	x	x	x	x		
ASAS	x	x	x	x		
FIREFINDER			x	x	x	x

d. Association with Requirements Documents. ROC for most battlefield electronic systems specify the performance parameters of the system. The development of these test methods and procedures will permit measurement of the software performance to determine if the specified requirements have been satisfied.

10. RESOURCES

a. Financial

(1) Funding Breakdown

	Dollars (Thousands) FY 85	
	In-House	Out-of-House
Personnel Compensation	6.0	
Travel	1.0	
Contractual Support		52.0
Material & Supplies	1.0	
Subtotals	8.0	52.0
FY Totals	60.0	

Explanation of Cost Categories

(a) Personnel Compensation. Cover in-house labor costs for the principal investigator and other in-house project support personnel.

(b) Travel. Travel is required to coordinate the tasks with other government agencies.

(c) Contractual Support. Approximately 90 percent of the work will be accomplished by a contractor under an existing service contract.

(d) Materials and Supplies. Incidental supplies will be required to support the investigation.

b. Anticipated Delays. No delays are anticipated at this time.

c. Obligation Plan (FY 85)

Obligation Rate (Thousands)	FQ 1	2	3	4	Total
	54.0	2.0	2.0	2.0	60.0

Software Testing of Multiprocessor Systems (cont)

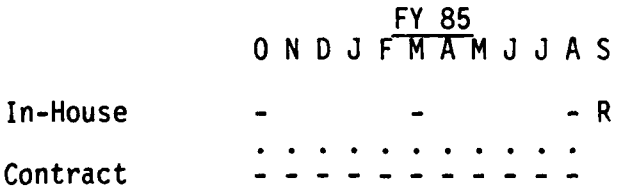
d. In-House Personnel

(1) Requirements

Type	Number	FY 85 Manhours Required	FY 85 Manhours Available
Electronic Engr GS-0855	1	300	300

(2) Resolution of Non-available Personnel. Not applicable.

11. INVESTIGATION SCHEDULE.



Symbols:
- - - Active investigation (all categories)
. . . Contract monitoring (in-house only)
R Final report at HQ TECOM

12. ASSOCIATION WITH TOP PROGRAM. It is expected that this investigation will result in a new TOP.

FOR THE COMMANDER:

(signed)
MELVIN FOWLER
LTC, SigC
Director of Materiel Test

(BLANK PAGE)

APPENDIX B

ACRONYMS

(BLANK PAGE)

ASAS.....All Source Analysis System
BAS.....Battlefield Automated Systems
BIT.....Built-In Test
C^3I.....Command, Control, Communications, and Intelligence
CPU.....Central Processing Unit
CRT.....Cathode Ray Tube
DC^3I.....Distributed C^3I
DT.....Developmental Test
DTSS.....Digital Topographic Support System
FIDS.....Facility Intrusion Detection System
FY.....Fiscal Year
GPIB.....General Purpose Interface Bus
GPS.....(see NAVSTAR GPS)
HM.....Hybrid Monitor
ICE.....In-Circuit Emulator
I/O.....Input/Output
JINTACCS.....Joint Interoperability of Tactical Command and Control Systems
JTIDS.....Joint Tactical Information Distribution System
LAN.....Local Area Network
MCS.....Maneuver Control System
MOP.....Measure of Performance
NAVSTAR GPS...Global Positioning System
NBS.....National Bureau of Standards
PCOTES.....Prototype Carrier Operational Test and Evaluation Site
PJH.....PLRS/JTIDS Hybrid
PLRS.....Position Location Reporting System
PM.....Program Manager
PROM.....Programmable Real-Only Memory
RAID.....Remote Access Interactive Debugger
ROC.....Required Operational Capability
ROM.....Read-Only Memory
RPV.....Remotely Piloted Vehicle
SHORAD C^2.....Short-Range Air Defense, Command and Control
SIGMA.....Force-Level Maneuver Control
STMS.....Software Testing of Multiprocessor Systems
SUT.....System Under Test

TCAC-(D).....Technical Control and Analysis Center (Division)
TECOM.....U.S. Army Test and Evaluation Command
TIS.....Test Item Stimulator
TOP.....Test Operations Procedure
TRMS.....Test Resource Management System
USAEPG.....U.S. Army Electronic Proving Ground
VAX.....Virtual Address Extension
VISTA.....Very Intelligent Surveillance and Target Acquisition System
V&V.....Verification and Validation

APPENDIX C

HARDWARE/SOFTWARE MONITOR COMPARISON

(BLANK PAGE)

1.0 INTRODUCTION

a. This appendix provides a brief comparison of hardware and software monitors, and provides arguments to show the additional capability inherent in software monitors.

b. The introduction provides an overview of the content of the other sections. The second section addresses problems encountered when the monitoring tools are interfaced with a new SUT. The third section deals with testing problems encountered after the SUT has been interfaced with the monitoring tools and the actual testing is underway. Problems are considered in an environment where both the hardware monitors and the software monitors have complete access to the SUT from which they are required to extract test data. This approach is taken so that neither monitor type is given an unfair advantage. In all sections, references to the capabilities of the hardware monitor are based on features found on most monitors.

c. Any monitoring tool which can be expected to provide meaningful test data from a multiprocessor system (especially a networked multiprocessor system) must have a central monitor processor which correlates incoming test data from the various SUT processors and presents or stores this data. Further discussion assumes such a central monitor is required whether hardware monitors or software monitors are used to fulfill testing requirements.

d. It should be recognized that whether a hardware monitoring, software monitoring, or HM approach is chosen, interfacing the existing monitor to a new SUT will most likely require new hardware and software components. This does not imply that what was developed as the basic monitoring tool will need a complete redesign. The basic monitoring tool provides the necessary framework, which is tailored to the varying requirements imposed by a new SUT. This tailoring for the hardware monitors may include the purchase of new bus interface or emulation cards. For software monitors, this tailoring may include the implementation of a new software kernel.

e. It should be noted that no one monitoring tool will address any system for testing without some tailoring to the specific system. Alterations will be required to fit any monitoring tool to a particular SUT configuration and physical interfaces. The key to addressing these varying systems is to have a flexible monitoring tool which has been designed with the ease of adaptation to new SUT configurations as a major concern.

2.0 INTERFACE PROBLEMS

This section provides an insight into problems which may arise during the initial interfacing of the monitoring tool with a new SUT. Problems presented here tend to justify the existence of software monitors. This is by no means a complete list of the problems which may be encountered during the interfacing effort.

2.1 Hardware monitors require access to the internals of the processors which are to be monitored. Gaining this access can present several problems.

a. If the processor enclosure cannot be opened during testing, the hardware monitor can only be used to detect signals which are present outside the enclosure (e.g., I/O line signals).

b. If the processor internals have been sealed (e.g., coated with epoxy) to withstand severe environments, it may not be possible to connect the hardware monitor's probes to the required test points.

c. If the physical constraints of the processor enclosure disallow the insertion of test probes, emulators, or bus interface cards, a hardware monitor will not have the required access. This problem can sometimes be eliminated through the use of extender cards; however, the introduction of these extender cards into the system may itself cause problems. (These cards increase the length of the path over which the various signals must flow and can actually induce timing problems.)

2.2 A multiprocessor system which contains several like processors presents an interesting cost comparison problem.

a. When using hardware monitors to test a SUT which contains several processors (alike or not), it is necessary to have one hardware monitor for each processor (or at least, one hardware monitor for each pair of processors). Additionally, a bus interface card, emulator, or probe set must be provided for each processor in the SUT.

b. When using software monitors to address several like processors, generally only one basic software kernel is developed. Copies of this kernel are placed in each of the like processors. However, if conventional I/O facilities are used by the software monitors to relay their test data to the central monitor, it may be necessary to provide a hardware protocol converter (I/O interface unit) for each of the SUT processors.

2.3 The introduction of a software kernel into the existing program structure of a SUT processor can cause problems.

a. In some cases, it may be impossible to insert a software kernel into a SUT processor's program structure. Such cases would require the use of a hardware monitor.

b. If emulation is possible (generally a microprocessor application), an interesting situation develops. Most emulators allow the use of "extended" memory. This is memory belonging to the emulator, itself, and not to the SUT computer. Memory maps can be defined within the emulator which force references to the SUT processor's memory to actually read/write the emulators extended memory. A normal application of this capability is to:

. Copy a section of the SUT processor's programmable read only memory (PROM) into the emulator's extended memory.

. Define a memory map within the emulator which will cause references to that section of SUT memory to access the emulator's extended memory copy.

. Make alterations (patches) to the copy residing in extented memory.

. Execute the code to see if the patches produce the desired results. If so, a new PROM can be made which reflects these patches.

This, in effect, allows PROM to be temporarily altered to suit a user's needs.

c. The interesting situation which develops in a testing environment using an emulator's extended memory can be explained using the following example:

(1) A software kernel is developed and downloaded into the emulator's extended memory.

(2) The kernel is "hooked" into the SUT processor's software structure by:

(a) Copying small sections of SUT code into the emulator's extended memory.

(b) Inserting interrupts or jump instructions into the copied code so that control can be passed to the software kernel when an event occurs.

(c) Defining a memory map such that references to the copied sections of code will be satisfied using the emulator's extended memory, instead of the SUT computer's memory.

(3) The test is run.

(a) Code is executed from the SUT computer's memory until references are made to instructions contained in one of the copied sections of code. These references are satisfied using the emulator's extended memory.

(b) Normal SUT processing continues until one of the inserted jumps or interrupts which define an event is executed. At this time, software kernel processing takes place.

(c) The software kernel is executed by the SUT processor from the emulator's extended memory. Upon software kernel processing completion, normal SUT processing continues until another inserted jump or interrupt is executed.

This method of monitoring is, in effect, HM. An added benefit is that the software monitor has shifted one of its resource requirements (namely memory) from the SUT computer to the hardware monitor. In this instance, the distinction between hardware monitoring and software monitoring has become blurred. A software monitor is present. That is, a program has been embedded in the SUT processor (or in this case "pseudo-embedded") which takes full advantage of that processor's instruction set for the purpose of gathering and relating test data. However, this software kernel actually resides within the hardware monitor's emulator memory and relies heavily upon that device for software monitor interaction with the SUT computer. In this configuration, the two monitor types have become inseparable.

3.0 TESTING PROBLEMS

This section presents problems which can arise during the testing of a system that cannot be sufficiently resolved using a hardware monitor.

Examples of these problems are provided with each problem cited. The basic assumption in this section is that both types of monitors (hardware and software) have the access to the SUT which they require to extract test data.

3.1 A MOP requires arithmetic functions be performed real-time to determine if an event has occurred. Hardware monitors do not provide a method to easily perform arithmetic operations on observed data.

Example A: Addressing a memory fragmentation problem requires that the amount of memory allocated and the amount of memory free be determined and the physical configuration be assessed to compute the extent of fragmentation. Collection of this data requires arithmetic functions be performed.

Example B: Queue monitoring sometimes requires that arithmetic functions be performed. This is the case if linked list queues are being traversed to determine (or verify) their length. (Each entry traversed must be indicated by adding one to a length counter.)

Example C: In observing the periodicity of a module's execution, it is often necessary to compute the delta time between executions.

Example D: Observing floating point values can greatly compound the problem if arithmetic functions must be performed on these values in a monitor processor which does not use the same representation for floating point values. To convert from one to another processor's representation requires determination of several factors, including: number of bits in the exponent and in the mantissa; whether the exponent represents a hex, octal, or binary shift count; sign conventions; exponent bias; and whether there is a phantom bit in the mantissa (highest order mantissa bit is understood to be a one, but a one is not present in the internal representation of the number).

3.2 A MOP requires comparison of values which are dynamic. A hardware monitor does not provide a method to compare values of this nature.

Example A: A value must be tracked and verified to be within limits which are influenced by other system variables. RPV's altitude and velocity must be compared with its pitch to determine if a crash is imminent.

Example B: One value must be verified to be functionally associated with another.

A>100.and.A=B

While a hardware monitor can determine if A>100 (assuming A is on a bus when the comparison needs to be made), it cannot easily determine if A=B (even if both are accessible on the bus when required).

3.3 A MOP requires active correlation of several memory locations that may not be present on a system bus when the correlation must be made. A hardware monitor is passive, and as such does not itself make memory requests but simply observes traffic on system busses.

Example A: Queue monitoring to determine if overflow conditions are imminent may require investigation of other queue's counters when an entry is added to a queue. (For example, queue A has higher priority than queue B;

their activity is monitored to determine if A processing is excessively blocking B's processing.)

Example B: Gathering of queue entries on an overflow condition may involve "indirect" addressing if a queue contains only pointers to the actual data packets.

3.4 A MOP requires the tracking of structures that are not resident in memory. A hardware monitor cannot dynamically alter its address monitoring ranges based on a SUT processor's activity.

Example A: A module that is rolled in and out of memory and is not assured of executing out of the same memory address range each time would require a software monitor to track it (via "hooks" to the executive program).

Example B: A data segment which is not kept resident in SUT processor memory would require a software monitor to determine when that segment is in memory (and possibly where) so that the required data can be extracted.

3.5 A MOP exceeds the event detection capacity of a hardware monitor.

Example A: Event detection requires inspection of register resident data at a certain point in a program's execution (e.g., inspection of an index value manipulated in a register).

Example B: A single event is determined by the occurrence of more than fifteen distinct happenings.

Example C: The definition of all of the events to be monitored exceeds the limit of fifteen distinct occurrences. (For example, one event requires six distinct happenings to define, another event requires five distinct happenings to define, and the last event to be monitored requires five distinct happenings. In this example, the last event to be monitored would exceed the limit of fifteen happenings available on some hardware monitors and could not be included in a single test run.)

(BLANK PAGE)

APPENDIX D

PROTOTYPE HARDWARE MONITOR CHARACTERISTICS

(BLANK PAGE)

1.0 TEKTRONIX DAS 9100

a. The Tektronix 9100 series digital analysis system consists of the color 9129 mainframe, optional data acquisition modules, and pattern generator modules. It can be remotely controlled via an RS-232 or General Purpose Interface Bus (GPIB) interface. Exact characteristics are dependent upon the options and configuration in use; Tektronix manuals should be referenced for the capability of a particular configuration.

b. The data acquisition modules enable up to 96 channels of information to be monitored and the data stored in a buffer memory with a depth of up to 4096 bits. The data acquisition module, which permits disassembly of microprocessor code (91A24 series), has a maximum sampling rate of 10 MHz and a buffer memory depth of 1024 bits. Trigger qualifiers are programmable, and accept up to three external inputs.

c. Many personality modules are available for the disassembly of microprocessor and minicomputer code. Probes are provided to monitor address and data busses. The 9100 does not provide emulation, however, this function is provided by other Tektronix products.

d. Precision event tracing and capture is provided through five independent word recognizers using two operational modes. The word recognizers provide up to 16 levels of sequential event tracing plus data qualification for selective data storage. In one mode of operation, data qualifier words are available for starting and stopping data storage. The 16-level sequential stack is available for trigger tracing of qualified data. Also, the occurrence counter, trace triggers counter, and 100 ns timer are available with the stack; and a SYNC output signal can be enabled at each level. Also, a parallel "OR" trigger RESET event recognizer is available for resetting the stack level to level one.

e. The second mode provides four word recognizers for logical data qualifier combinations, two word recognizers for enabling data storage and two word recognizers for disabling data storage. The 16-level sequential stack is also available for trigger tracing of qualified data.

f. In addition, two event counters and a 100 ns resolution timer combine with triggering to help track software in complex digital systems. Events can be counted up to 4096 occurrences per level and then reset if necessary. At each sequential trigger level a SYNC pulse out can be programmed to trigger or strobe external equipment.

g. An External Trigger Enable input allows the 91A24 modules to be automatically "armed" from an external source to look for a trigger on incoming data only when requested.

h. The 91A24 modules can be used to track down intermittent faults with the auto comparison between reference memory and the 91A24's data acquisition memory. And data differences can be displayed in color with user-defined mnemonic disassembly to help identify anomalies. Also provided is the ability to perform column masking in conjunction with a programmable compare window.

i. Up to 80 channels of programmable output are available with the 25 MHz pattern generator module. With output clocks, up to 10 separately programmable strobes are available. Subroutine nesting up to 16 levels, external

interrupts, single-key functions, and selectable Radix are featured in the pattern generator module.

j. The 9100 is equipped with a 160K byte cartridge tape drive. Remote and master/slave operation is provided by an RS-232 interface. All instrument status, including both menu information and acquisition and pattern generator memory contents can be obtained via this data link.

APPENDIX E

MEASURES OF PERFORMANCE

(BLANK PAGE)

1.0 MEASURES OF PERFORMANCE

1.1 Introduction

Many aspects of a system's performance may be analyzed. This section is concerned with those measures associated with verification of system performance. The MOPs described herein have been divided into five basic categories, as delineated in table I.

1.2 System Control

The first category, system control, consists of those aspects related to the overall control of the system. Most systems employ some mix of tables, queues, timers, and counters to control their processing both at the executive and application levels. It is important for the tester to be able to observe the activity and correlate these parameters to determine if system performance is within specifications.

1.2.1 Overhead

This is a measure of the percentage of CPU processing time required by the executive program to control a processor's applications programs. Examples of executive processing which constitute overhead are task scheduling, memory allocation, and message queuing. Measuring overhead provides one means for the tester to determine if adequate CPU resources are available for the application.

1.2.2 Queue Usage

a. This measurement provides a means of evaluating the management of system queues. It also provides a method of evaluating the effect of loading on the various queues and their effect on system performance.

b. Queue usage is a necessary measure for the tester, as it can provide an immediate indication that data is being lost. The tester can also use this measure to determine if queued data is being processed at the required rate. Additionally, it can be determined if queues have been improperly prioritized or have excessive/insufficient allocated memory (e.g., a queue allocated 10 locations is never observed to use more than one). This measure is normally analyzed in an effort by the testing organization to confirm compliance with specifications.

1.2.3 Throughput

a. This measurement identifies the amount of processing completed per unit of time at a system level.

b. A measurement of this nature is required to determine the overall impact of different loads on the system's performance. Throughput provides an overview from which conclusions can be drawn as to which functions are adversely affecting the system. This measure provides the higher level overview necessary to direct testing efforts if a problem is encountered.

Table I. Measures of Performance

Measures of Performance		VV	IT	HFL	SD	DT
SYSTEM CONTROL						
Overhead		X	X			X
Queue Usage		X	X		X	X
Throughput		X	X			X
MEMORY UTILIZATION						
Memory Fragmentation		X	X		X	X
Virtual Memory Overhead		X	X	X		X
COMPONENT USAGE						
Component Overlap		X	X	X		X
Component Response (including interrupts)		X	X	X		X
Component Utilization		X	X	X		X
SOFTWARE FUNCTIONS						
Function Trace		X	X	X	X	X
Function Execution Ratios		X	X		X	X
Function Periodicity		X	X		X	X
Function Duration		X	X		X	X
Function Processing Time		X	X		X	X
Function Overlap		X	X		X	X

VV = Verification/Validation
IT = Integration Testing
HFL = Hardware Fault Location
SD = Software Debug
DT = Developmental Testing

Table I. Measures of Performance (continued)

Measures of Performance		VV	IT	HFL	SD	DT
NETWORK COMMUNICATION						
Host Communication Matrix		X	X		X	X
Packet Type		X	X		X	X
Packet Size		X	X		X	X
Communications Throughput		X	X		X	X
Packet Interarrival Time		X	X		X	X
Channel Acquisition		X	X		X	X
Communication Delay		X	X		X	X
Collision Count		X	X		X	X
VV = Verification/Validation IT = Integration Testing HFL = Hardware Fault Location SD = Software Debug DT = Developmental Testing						

1.3 Memory Utilization

a. Many memory storage management schemes are in existence today. Most of these schemes fall into one of the seven following types:

- Single contiguous memory.
- Partitioned contiguous memory.
- Relocated partitioned contiguous memory.
- Paged memory.
- Demand-paged memory.
- Segmented memory.
- Segmented and demand-paged memory.

b. The MOPs in this category are designed to measure those system parameters associated with memory management.

1.3.1 Memory Fragmentation

Memory fragmentation is the result of allocating and deallocating dynamic storage space in a manner such that larger contiguous blocks of memory are not available when required. This means that at a given time there may be enough total free dynamic memory to support a memory request but only as several small non-contiguous blocks. If this occurs, a requester must wait until a large enough contiguous block becomes available. This condition can seriously impact system performance and therefore warrants monitoring by the tester.

1.3.2 Virtual Memory Overhead

Program execution time can be adversely affected by the utilization of virtual memory management. The amount of time spent in such activities as swapping working sets to secondary storage add to the system overhead and actual program execution time. The amount of time allocated to management of virtual memory can be measured to see if this time is excessive or whether certain software functions should be permanently resident.

1.4 Component Usage

A component or resource, as used in this context, is a physical entity within a computer system. It may include a CPU, I/O controller, tape drive, disk, etc. In particular, in a multiprocessor network environment, it most certainly includes other processors. That is, any one processor may be treated as a component of the system.

1.4.1 Component Overlap

a. This is a measure of the percent of time system components operate in parallel.

b. This measure is useful in a test environment because it allows the detection of blockage between two or more components of a system. Operation of a system component is sometimes specified in terms of non-interference with other components. Adherence to specifications can be confirmed using this measure. The tester can further use this measure to determine probable system bottlenecks by identifying non-overlapping components and system load scenarios which induce non-overlap conditions.

1.4.2 Component Response

a. This measure involves determining the amount of time required to complete processing of requests by system components.

b. This measure is important to the tester to insure that components respond within the specified time. The tester must also be concerned with the effects of multiple processes and peripheral contention in a multiprocessor environment. Varying load testing may be performed to accurately determine saturation points of the various components of a system. This type of testing is also necessary to evaluate the effects of response times that are within specifications but are slower than the unloaded nominal response time.

c. While component response would normally be used by the tester in the context of peripheral or intercomputer responses, it is also used to address intracomputer component response times. Intracomputer response times include the time required by a CPU to respond to an interrupt issued by another component. It is important for the tester to be able to address these intracomputer response times to determine if the method used to process these interrupts is inducing excessive delays.

1.4.3 Component Utilization

a. This measure is employed to evaluate the usage and resource requirements of components in a system.

b. An accurate measure of CPU time must be obtained to determine if the specified reserve CPU capacity has been met. In a multiprocessor environment, this measure can be used to determine an imbalance in the workload. This would allow the tester to suggest shifting of this load to bring a system's performance within specification.

c. The tester must also determine if a peripheral is performing within specification or if the utilization of a peripheral is impacting system performance beyond acceptable limits. This measure can additionally be used to suggest corrective action if the underuse of a peripheral indicates that part of its processing time or buffer memory can be allocated to other processes.

d. Component utilization can also be used to determine if adequate data transfer reserves are being maintained. This applies to the update rate of files, interprocessor communications, and to data transfer loads on I/O controllers.

1.5 Software Functions

This section addresses those measures related to software functions. A function can be defined as a single instruction, a short sequence of instructions, a module (procedure), a set of modules (task), or any combination of these which define some unique system processing. Within a multiprocessor system, this sequence can span processors for those functions which require interprocessor interactions.

1.5.1 Function Trace

a. This measure allows the determination of correctness by showing that an input data item has followed an expected path through the system and has produced the expected output. Additionally, results are used to indicate which software elements have been executed.

b. The tester is concerned with the ability to prove the correctness (or incorrect performance) of a SUT. A major component of correctness in computer systems is the production of an expected output from the application of a known input. A method often employed to prove correctness is the tracking of an input through the system, inspecting any pertinent intermediate results, and confirming that the expected output is produced. This is sometimes referred to as data flow analysis.

c. Another use of function tracing is to provide a history of the control flow. This information allows the tester to confirm that proper segments of a program are executed in the correct order. Depending upon the level of tracing, various levels of test coverage (or thoroughness) may be derived from function tracing. This tracing gives the tester a quantitative measure of the completeness of testing.

1.5.2 Function Execution Ratios

a. This measure uses the occurrence of of a function correlated with time or inputs to determine if a function is executing at the required rate.

b. This measure is essential to the tester to confirm that all processing required is being performed in a timely fashion. This measure can also be used in conjunction with saturation testing to confirm that nonessential functions are being suspended to allow critical functions to be completed (i.e., priorities are correctly assigned).

c. Since functions can be single instructions, this MOP could be used for an instruction mix analysis. Such an analysis is frequently helpful in uncovering excessive use of time-consuming instructions. By examining operation codes, frequency of instructions in a particular segment can be tallied for comparisons.

1.5.3 Function Periodicity

This measure, which provides information pertaining to the differential time between executions of a function, is important to the tester to determine if time-critical periodic processing is being performed at the required intervals. It is important that a monitor be able to extract, correlate, and present this information so that the tester can evaluate the influence of loading and other factors on this measure.

1.5.4 Function Duration

a. This measure is the total time required to complete a software function.

b. It is important the tester evaluate the response of the system as it is subjected to various loads. As an example, a tracking function could perform as specified in a light load scenario, but may be producing erroneous outputs under a heavier load. If this tracking function was defined by several modules, errors may not be detectable at the module level, as each module could complete processing within its specified time. The interference induced by increased executive processing (overhead) required by heavier loads could result in scheduling delays of modules which comprised the entire tracking function. In this case, the problem could be quickly identified by function duration time measurements.

1.5.5 Function Processing Time

a. This is a measure of the CPU time required by a function to complete its processing. This measure excludes the executive processing time (overhead) required to control the processing of a function.

b. Function processing time supplements the information available from an analysis of function duration. This measure can be used to determine the degree to which total function time consists of individual module CPU execution times.

1.5.6 Function Overlap

a. This measure correlates concurrent software functions.

b. This measure can be used to determine if functions are blocking one another, resulting in possible deadlock conditions, or if functions which should execute in sequence are properly synchronized.

1.6 Network Communication

The network communication MOPs were derived from an examination of the National Bureau of Standards (NBS) monitor. These MOPs are applicable to multiprocessor configurations where LANs provide the communication paths among processors. Because of the overlap between communication link monitors and digital message test drivers, such as the TIS, these MOPs were not fully analyzed but are presented below for completeness.

1.6.1 Host Communication Matrix

This measure indicates the traffic flow between connected components of the SUT.

1.6.2 Packet Type

This measure indicates the distribution of each type of packet transmitted.

1.6.3 Packet Size

This measure records the number and proportion of data packets of particular length classes.

1.6.4 Communications Throughput

The communications throughput measure determines the actual rate of message transfer on the communications channels. Also included is the channel capacity used for overhead and for retransmission of messages received in error. Other related statistics on message error rate, bit error rate, channel availability, etc., would be available for analysis as a result of the data elements required to derive the communications throughput measure.

1.6.5 Packet Interarrival Time

This measure indicates the number of packet interarrival times which fall into particular time classes. An interarrival time is the time between consecutive carrier (network busy) signals.

1.6.6 Channel Acquisition Delay

This measure records the times spent by components contending for and acquiring the channel. A channel acquisition delay begins when a component becomes ready to transmit a packet and ends when its first bit is transmitted onto the channel. Included is all of the time spent deferring due to a busy channel and the time recovering and backing off from one or more collisions.

1.6.7 Communication Delay

This measure indicates the delays that components incur in communicating packets to their destinations. A communication delay begins when an original packet becomes ready for transmission and ends when that packet is received by the destination (which may be several transmissions later). One communication delay exists for each packet communicated. This delay may include several channel acquisition delays.

1.6.8 Collision Count

This measure tabulates the number of collisions a packet of any type encounters before completion of a successful transmission.

APPENDIX F

HYBRID MONITOR METHODOLOGY

(BLANK PAGE)

1.0 INTRODUCTION

This appendix describes the development of the HM methodology, which led to the concept of integrating the capabilities of both hardware and software monitors controlled by a central controller--the HM approach--in an attempt to exploit the desirable features of both monitors while minimizing their shortcomings.

1.1 Hybrid Monitor Methodology

The HM methodology progressed from the basic test methodology to an analysis of the tools suitable for multiprocessor testing. An analysis of the tool features was used to select the most viable approach to meet the investigation objectives.

1.1.1 Test Methodology

a. A method for approaching the testing of software is a performance analysis. A performance analysis defines those characteristics of the system which can be used to determine correctness of processing. These characteristics include: the responsiveness of the system under various loads, the concurrency of processes within the system, and the capacity of the system.

b. These characteristics can be determined by obtaining MOPs from the SUT. These MOPs were addressed in appendix E. It should be noted that the MOPs intentionally overlap one another. To define MOPs in terms of strict non-overlapping measures is not possible if the measures are to be applied to a variety of systems and implementations.

c. In order to extract the measures, a testing organization must have tools at its disposal which are capable of performing the measurements which the tester deems necessary to prove correctness.

d. The most widely used tools for performing these measurements are hardware monitors and software monitors. A newer approach, the HM, is potentially a flexible tool capable of gathering a wide variety of data. These tools are described below with the methods of data collection used.

e. All of the MOPs address the resources of a system. The resources of a system are memory, processors, devices, and information. Memory refers to the high speed storage of computers. Memory types include read/write, ROM, and PROM. Processor types include CPUs and I/O controllers. Devices include disks, tape drives, and user terminals. Information refers to the non-hardware portion of a system. Information includes messages, files, data, applications programs, and executive programs.

f. The structural outline of a performance analysis is presented in figure 1.

1.1.2 Multiprocessor Software Monitor

a. A software monitor is a test tool which is inserted into the program structure of the SUT. In a multiprocessor environment, this type of monitor is generally implemented as an external processor (one which is not a part of the SUT) which communicates with kernels of code embedded in each of the

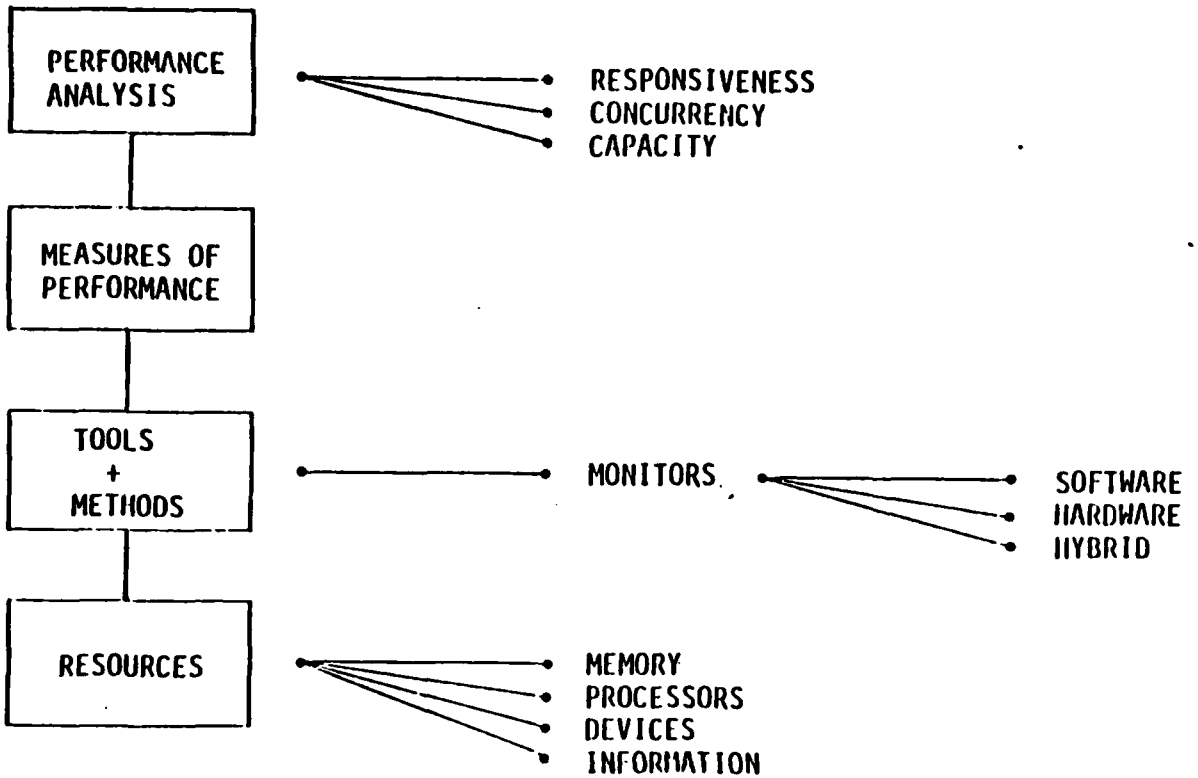


Figure 1. Performance Analysis Methodology

processing units which comprise the test system. These kernels accept commands from the test operator via I/O from the external processor. Commands are interpreted, and the requested data is gathered and sent back to the external computer via an I/O channel.

b. This test tool requires that the SUT be capable of supporting the kernel's resource requirements (memory, CPU time, I/O port, I/O handling). A tool of this type is intrusive, for it must use the resources being observed by the tester. Therefore, the tester must know the impact of the software monitor on the system.

c. An example of a multiprocessor configuration for a software monitor is shown in figure 2.

1.1.3 Multiprocessor Hardware Monitor

a. A hardware monitor is a device which is used to sense changes in the state of the hardware components that comprise a SUT. This sensing is carried out by attaching probes to the hardware at test points which provide signals indicating changes of state. These probes are used to monitor the address bus, data bus, I/O line signals, etc. These signals are correlated using timers, counters, and comparators. The data collected is formatted and presented in various displays or stored for post-processing.

b. A hardware monitor is generally non-intrusive, for it does not draw on any of the resources of the SUT. This type of monitor is limited by the accessibility of the required test points, the number of probes available to collect the data, and the complexity of defining the events to be monitored.

c. An example of a multiprocessor configuration for a hardware monitor is shown in figure 3.

d. A hardware emulator is a device commonly thought of as a hardware monitor but in actuality far exceeds a monitor's capability. The idea behind an emulator is that the CPU is replaced with a device that meets the same performance of the CPU, plus allows viewing the internal workings of the CPU. Such items as registers, flags, and instruction execution can be closely monitored in real time. When halted, this type of device permits changing registers and flags, thereby allowing a variety of reconfigurations of the program state.

e. With current technology, most emulators are associated with microprocessor applications. Under this application, they are commonly called in-circuit emulators (ICE). Most microprocessors are marketed with an associated ICE available for use in the software development and hardware/software integration. Use of hardware emulators outside of microprocessors is limited due to the enormous development cost. For example, a hardware emulator for a VAX would probably rival, if not exceed, the development cost of the VAX itself.

1.1.4 Multiprocessor Hybrid Monitor

a. A HM is the combination of a software and a hardware monitor which are used in unison to collect and correlate data from a SUT.

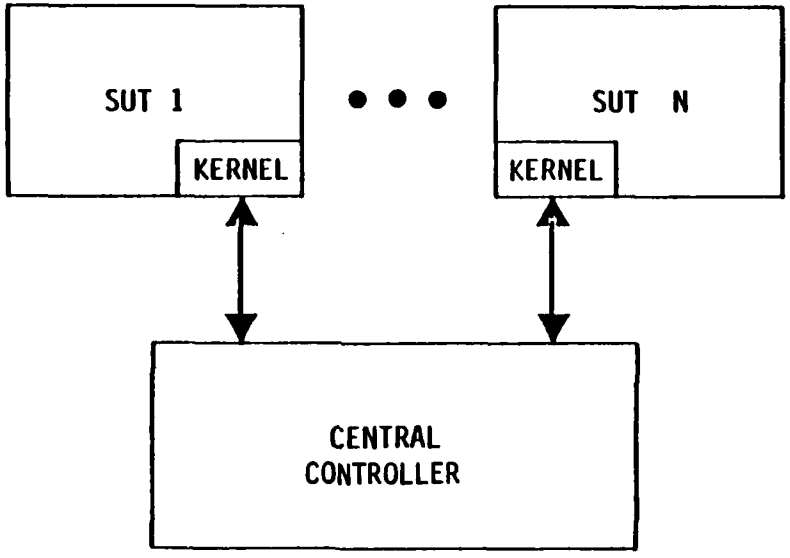


Figure 2. Software Monitor Configuration

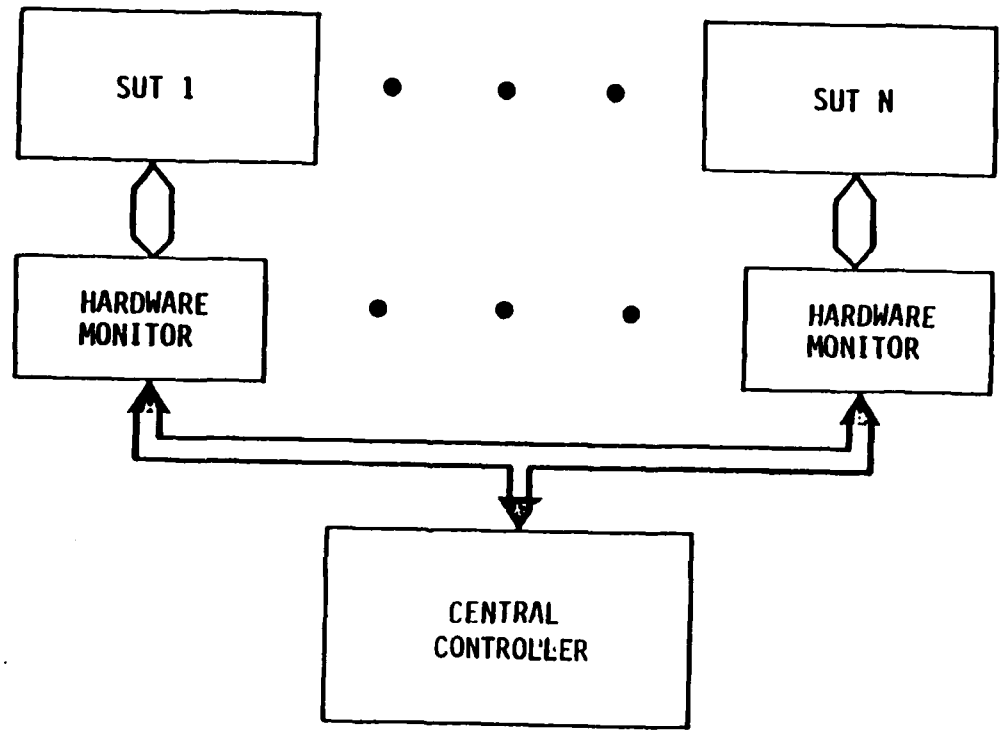


Figure 3. Hardware Monitor Configuration

b. As a hybrid contains both types of monitors, it can be applied to any test configuration which could be addressed by the individual monitors. The ability to blend the qualities of the two monitors also allows the hybrid to address test problems which could not be approached using either of the monitors individually.

c. A hybrid expands the event definition capabilities available to the tester by combining the abilities of both types of monitors. This combination also allows the elimination of some undesirable features of the individual monitors. Detecting certain events with software may remove a requirement to physically open a processor's container to allow access with hardware probes. Using a hardware monitor to detect the flagging of an event processed by a software monitor, greatly reduces the software monitor's impact on the test system as I/O is no longer required to relate occurrences to the external processor.

d. A test situation which required the complex logic capabilities of a software monitor and the high sampling rates attainable with a hardware monitor could be addressed using a hybrid as follows:

(1) The software monitor's kernel program is used to perform the complex logic.

(2) The result of the logic performed is stored into a memory word of the SUT.

(3) The hardware monitor is used to detect the storing of this memory word.

(4) The hardware monitor extracts this word when stored and processes it or passes it to the software monitor's external computer for processing.

(5) The hardware monitor is used, at the same time, to collect timing information on the software monitor's processing. This allows the latter's impact on the test system to be accurately assessed.

e. This blending of capabilities can span the full spectrum from exclusive use of the software monitor to exclusive use of the hardware monitor to allow the tester to extract the required data from the SUT.

f. An example of a multiprocessor configuration for a HM is shown in figure 4.

1.1.5 Comparison of Monitor Characteristics

a. A comparison of the three types of monitors was made based on characteristics considered desirable for a general purpose tool. The monitors were evaluated on their ability to display the characteristics over a wide variety of applications. The three monitor types (hybrid, software, and hardware) are shown in table II, with the results of the evaluation dichotomized into strong and weak.

b. Examination of the characteristics of the monitors shows clearly, in the hybrid approach, the synergy of an integrated software and hardware

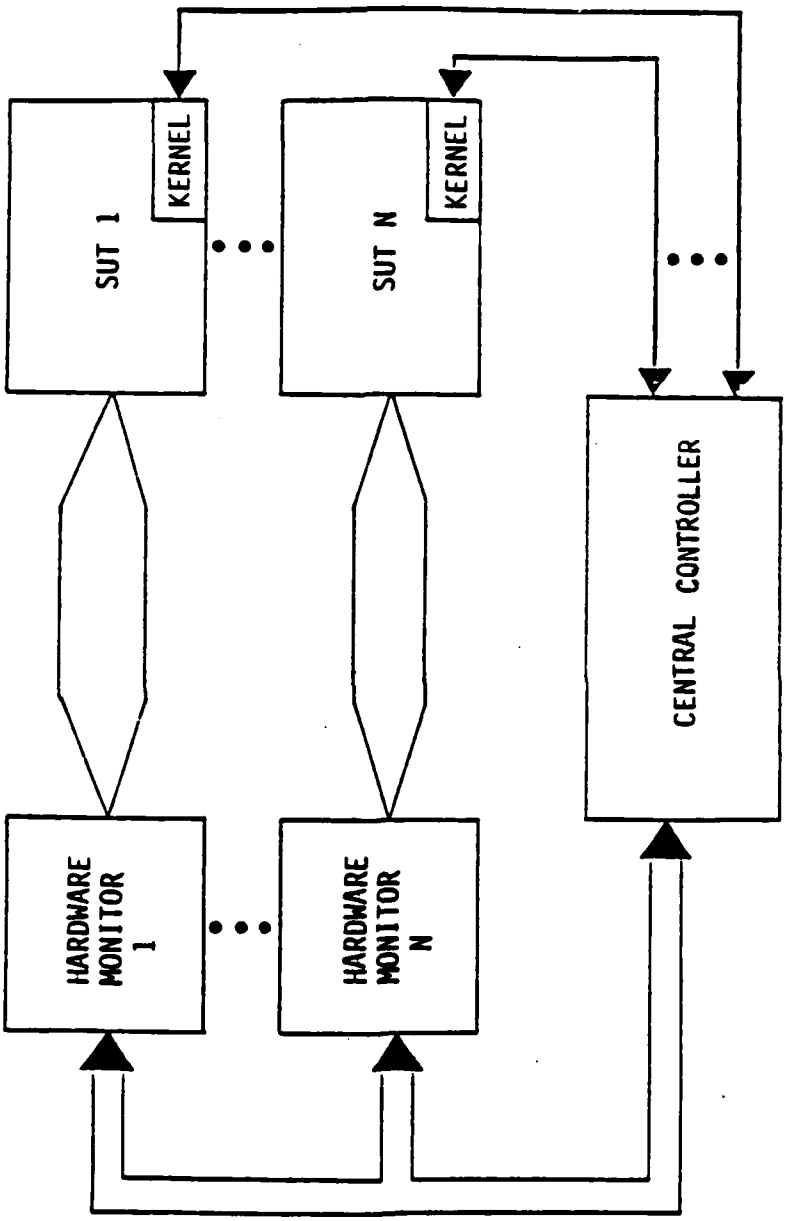


Figure 4. Hybrid Monitor Configuration

Table II. Comparison of Monitor Characteristics

HYBRID	SOFTWARE	HARDWARE	CHARACTERISTIC
S	S	W	Highly complex logic functions
S	W	S	Does not require test system resources
S	S	W	Able to control/be controlled by test systems
S	S	W	Access to any processor memory
S	W	S	Access to data which is not available through processor's instruction set
S	S	W	Able to monitor software not stationary in memory
S	W	S	High timer resolution/event rate
S	S	W	Able to correlate maximum number of inputs
W	W	S	Portability
S	W	S	Able to monitor events on external devices
W	W	W	Is useful without intimate knowledge of test system

S: Strong
W: Weak

architecture. The only drawbacks to a HM, for the features examined, are the lack of portability and the expert knowledge required. Because of the advantages that the hybrid approach offers over stand-alone software or hardware monitors, the HM concept was chosen for further design and development efforts.

(BLANK PAGE)

APPENDIX G

DISTRIBUTION

(BLANK PAGE)

DISTRIBUTION LIST

Addressee	Number of Copies
Commander U.S. Army Test and Evaluation Command ATTN: AMSTE-TC-M AMSTE-TO AMSTE-EV-S AMSTE-TE Aberdeen Proving Ground, MD 21005-5055	 3 2 1 6
Commander Defense Technical Information Center ATTN: DTIC-DDR Cameron Station Alexandria, VA 22314-5000	 2
Commander U.S. Army Aberdeen Proving Ground ATTN: STEAP-MT-M Aberdeen Proving Ground, MD 21005-5000	 2
Commander U.S. Army Yuma Proving Ground ATTN: STEYP-MSA Yuma, AZ 85364-5000	 2
Commander U.S. Army Jefferson Proving Ground ATTN: STEJP-TD-E Madison, IN 47250-5000	 1
Commander U.S. Army Dugway Proving Ground ATTN: STEDP-PO-P Dugway, UT 84022-5000	 1
Commander U.S. Army Cold Regions Test Center ATTN: STECR-TM APO Seattle, WA 98733-5000	 1
Commander U.S. Army Electronic Proving Ground ATTN: STEEP-TM-AC Fort Huachuca, AZ 85613-7110	 4

Addressee	Number of Copies
Commander U.S. Army Tropic Test Center ATTN: STETC-TD-AB APO Miami, FL 34004-5000	1
Commander U.S. Army White Sands Missile Range ATTN: STEWS-TE-PY STEWS-TE-O STEWS-TE-M STEWS-TE-A White Sands Missile Range, NM 88002-5000	4 1 1 1

DTIC

FILMED

4-86

END